Devang Patel Institute of Advance Technology and Research

DEPSTAR

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./ Mrs. Probin Bhagchandani

of 5CE1 *Class,*

ID. No. 22DCE006 *has satisfactorily completed*

his/ her term work in Matchine learning *for*

the ending in Nov. 2024/2025

Date : 16/10/24

K. S. Patel

Sign. of Faculty

Dgang

Head of Department

22DCE006-Probin Bhagchandani Practical-1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/customer_shopping_data.csv')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99457 entries, 0 to 99456
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   invoice_no      99457 non-null  object
 1   customer_id     99457 non-null  object
 2   gender          99457 non-null  object
 3   age             99457 non-null  int64
 4   category        99457 non-null  object
 5   quantity        99457 non-null  int64
 6   price           99457 non-null  float64
 7   payment_method  99457 non-null  object
 8   invoice_date    99457 non-null  object
 9   shopping_mall   99457 non-null  object
dtypes: float64(1), int64(2), object(7)
memory usage: 7.6+ MB
None
```

```
import numpy as np
blank_arr = np.zeros((3))
print("Blank array (zeros):\n", blank_arr)

predef_data = [2,4,6,8,10]
predef_arr = np.array(predef_data)
print("Array with predefined data:\n", predef_arr)

patt_arr = np.zeros((3,3), dtype=int)
patt_arr[1::3] = 7
print("Specific pattern array:\n", patt_arr)
```

```
Blank array (zeros):
 [0. 0. 0.]
Array with predefined data:
 [ 2  4  6  8 10]
Specific pattern array:
 [[0 0 0]
 [7 7 7]
 [0 0 0]]
```

```
import numpy as np
a = np.arange(10)
print("original array",a)
s = slice(2,7,2)
print("slice function",a[s])

b=np.array(a-2)
print("updated array",b)
```

```
original array [0 1 2 3 4 5 6 7 8 9]
slice function [2 4 6]
updated array [-2 -1  0  1  2  3  4  5  6  7]
```

```
arr = np.arange(12)

a = arr.reshape(3,4)

print('Original array is:')
print(a)

print('Modified array is:')

for x in np.nditer(a):
    print(x)
```

```
Original array is:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Modified array is:
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

```
import numpy as np
mydata = np.loadtxt("textfile.txt", dtype=int)
print(mydata)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
import pandas as pd
df=pd.read_csv('customer_shopping_data.csv')
```

```
df5=df.iloc[0:25]
df5
```

	invoice_no	customer_id	gender	age	category	quantity	price	payment_methoc
0	I138884	C241288	Female	28	Clothing	5	1500.40	Credit Carc
1	I317333	C111565	Male	21	Shoes	3	1800.51	Debit Carc
2	I127801	C266599	Male	20	Clothing	1	300.08	Casl
3	I173702	C988172	Female	66	Shoes	5	3000.85	Credit Carc
4	I337046	C189076	Female	53	Books	4	60.60	Casl
5	I227836	C657758	Female	28	Clothing	5	1500.40	Credit Carc
6	I121056	C151197	Female	49	Cosmetics	1	40.66	Casl
7	I293112	C176086	Female	32	Clothing	2	600.16	Credit Carc
8	I293455	C159642	Male	69	Clothing	3	900.24	Credit Carc
9	I326945	C283361	Female	60	Clothing	2	600.16	Credit Carc
10	I306368	C240286	Female	36	Food & Beverage	2	10.46	Casl
11	I139207	C191708	Female	29	Books	1	15.15	Credit Carc
12	I640508	C225330	Female	67	Toys	4	143.36	Debit Carc
13	I179802	C312861	Male	25	Clothing	2	600.16	Casl
14	I336189	C555402	Female	67	Clothing	2	600.16	Credit Carc
15	I688768	C362288	Male	24	Shoes	5	3000.85	Credit Carc
16	I294687	C300786	Male	65	Books	2	30.30	Debit Carc
17	I195744	C330667	Female	42	Food & Beverage	3	15.69	Credit Carc
18	I993048	C218149	Female	46	Clothing	2	600.16	Casl
19	I992454	C196845	Male	24	Toys	4	143.36	Casl
20	I183746	C220180	Male	23	Clothing	1	300.08	Credit Carc
21	I412481	C125696	Female	27	Food & Beverage	1	5.23	Casl

```
df.to_csv('customer_data.csv')
```

```
for i,j in df.iloc[:3].iterrows():
    print(i, j)
    print()
```

```
0 invoice_no           I138884
customer_id          C241288
gender                Female
age                       28
category            Clothing
quantity                   5
```

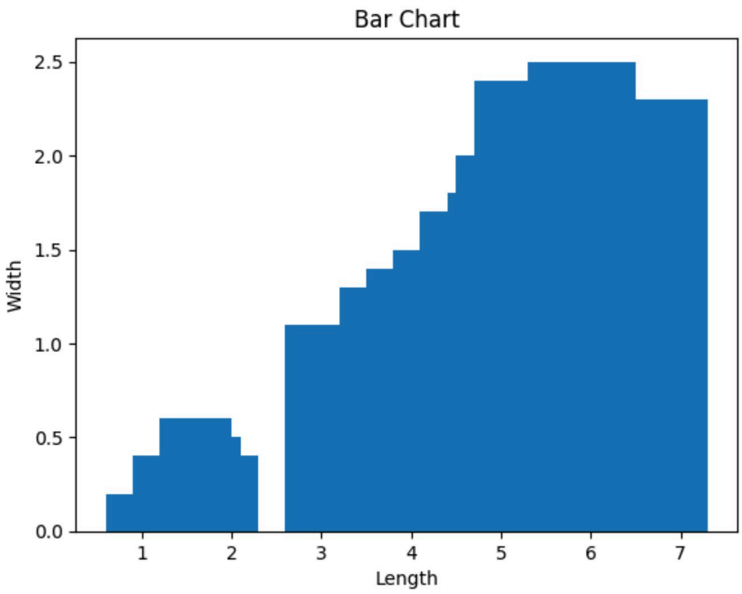
```
price                 1500.4
payment_method    Credit Card
invoice_date         5/8/2022
shopping_mall          Kanyon
Name: 0, dtype: object

1 invoice_no              I317333
customer_id              C111565
gender                      Male
age                           21
category                   Shoes
quantity                       3
price                    1800.51
payment_method       Debit Card
invoice_date         12/12/2021
shopping_mall     Forum Istanbul
Name: 1, dtype: object

2 invoice_no          I127801
customer_id         C266599
gender                 Male
age                      20
category           Clothing
quantity                  1
price                300.08
payment_method         Cash
invoice_date      9/11/2021
shopping_mall     Metrocity
Name: 2, dtype: object
```

```
cln=df.iloc[:3]
for i in cln:
  print(cln)
```

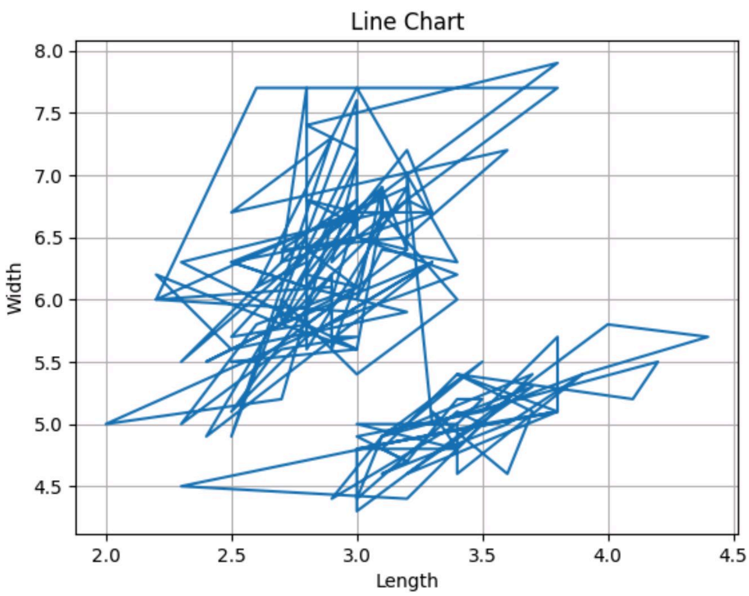
```
  payment_method invoice_date   shopping_mall
0    Credit Card     5/8/2022          Kanyon
1     Debit Card   12/12/2021  Forum Istanbul
2           Cash    9/11/2021       Metrocity
  invoice_no customer_id  gender  age  category  quantity    price  \
0    I138884     C241288  Female   28  Clothing         5  1500.40
1    I317333     C111565    Male   21     Shoes         3  1800.51
2    I127801     C266599    Male   20  Clothing         1   300.08

  payment_method invoice_date   shopping_mall
0    Credit Card     5/8/2022          Kanyon
1     Debit Card   12/12/2021  Forum Istanbul
2           Cash    9/11/2021       Metrocity
  invoice_no customer_id  gender  age  category  quantity    price  \
0    I138884     C241288  Female   28  Clothing         5  1500.40
1    I317333     C111565    Male   21     Shoes         3  1800.51
2    I127801     C266599    Male   20  Clothing         1   300.08

  payment_method invoice_date   shopping_mall
0    Credit Card     5/8/2022          Kanyon
1     Debit Card   12/12/2021  Forum Istanbul
2           Cash    9/11/2021       Metrocity
```

```
  payment_method invoice_date   shopping_mall
0    Credit Card     5/8/2022          Kanyon
1     Debit Card   12/12/2021  Forum Istanbul
2           Cash    9/11/2021       Metrocity
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/Iris.csv')
plt.bar(df['PetalLengthCm'],df['PetalWidthCm'])
plt.xlabel('Length')
plt.ylabel('Width')
plt.title('Bar Chart')
plt.show()
```

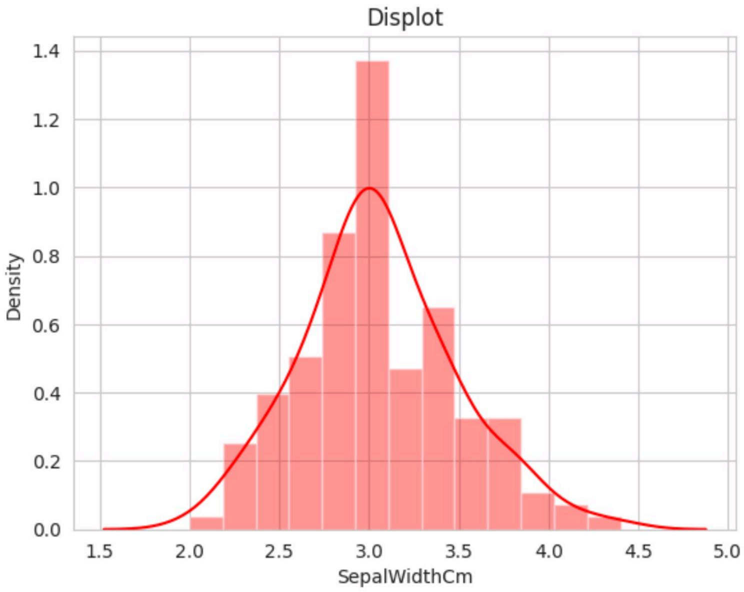



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/Iris.csv')
plt.plot(df["SepalWidthCm"], df["SepalLengthCm"])
plt.title("Line Chart")
plt.xlabel("Length")
plt.ylabel("Width")
plt.grid(True)
plt.show()
```

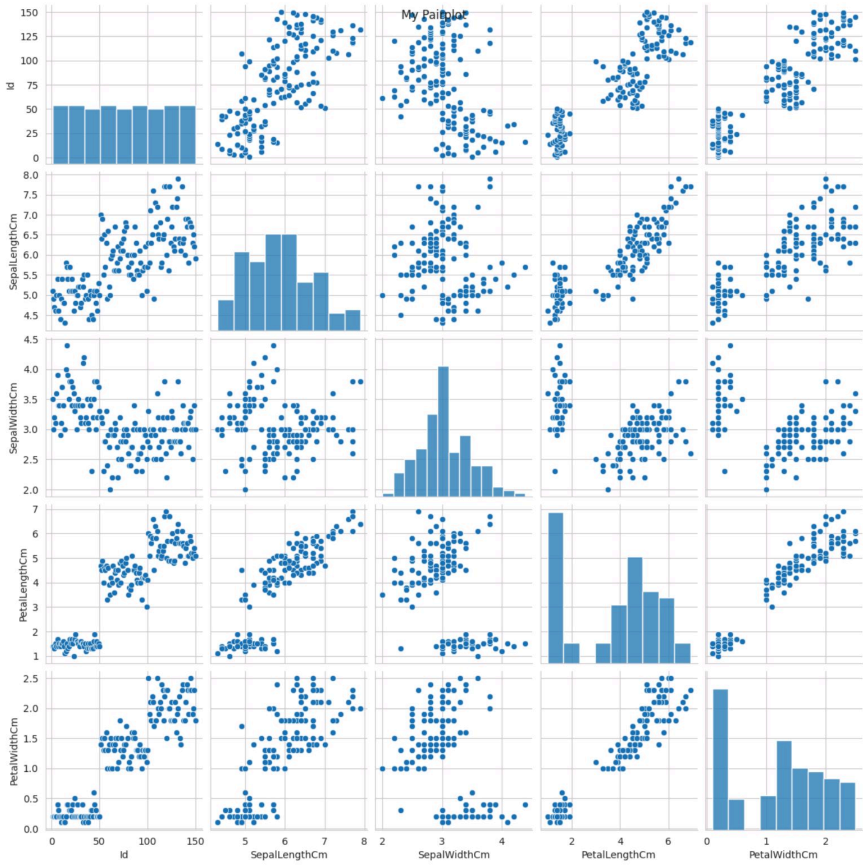


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/Iris.csv')
print("Displot")
sns.set_style('whitegrid')
sns.distplot(df['SepalWidthCm'], color ='red').set(title='Displot')
```

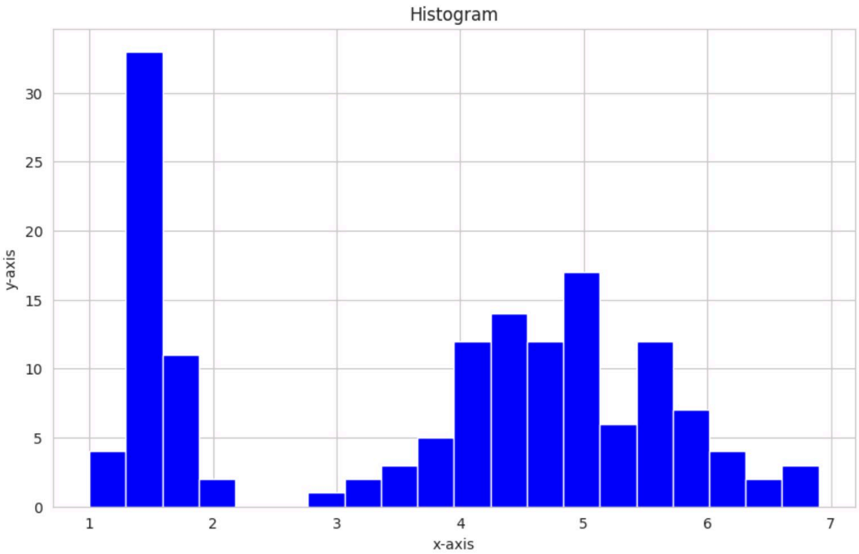
```
Displot
<ipython-input-38-303bdbfd5689>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['SepalWidthCm'], color ='red').set(title='Displot')
[Text(0.5, 1.0, 'Displot')]
```

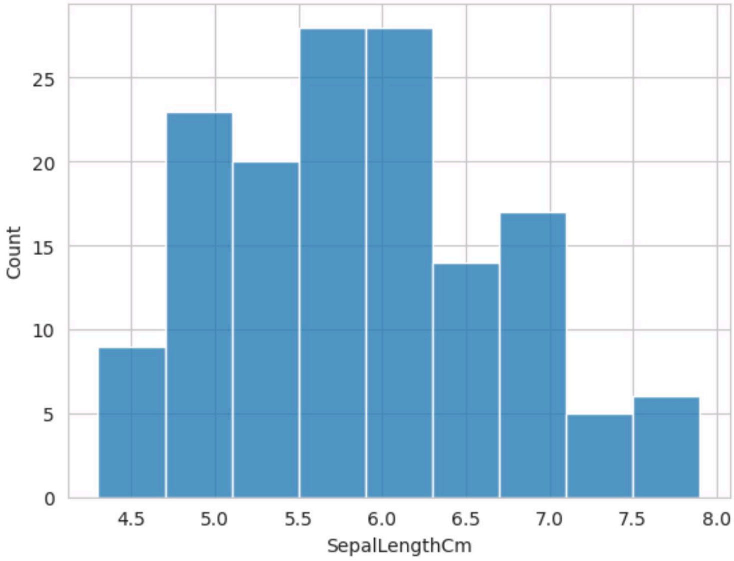


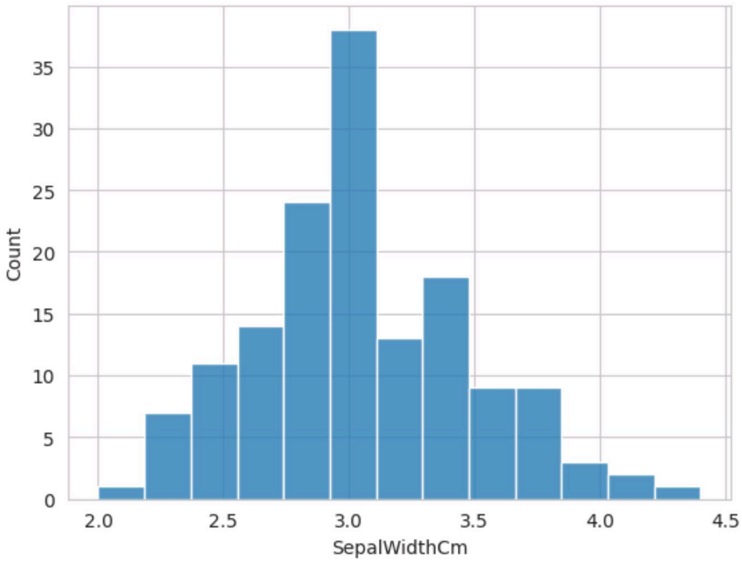

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/content/Iris.csv')
x=sns.pairplot(df)
x.fig.suptitle("My Pairplot")
plt.show()
```



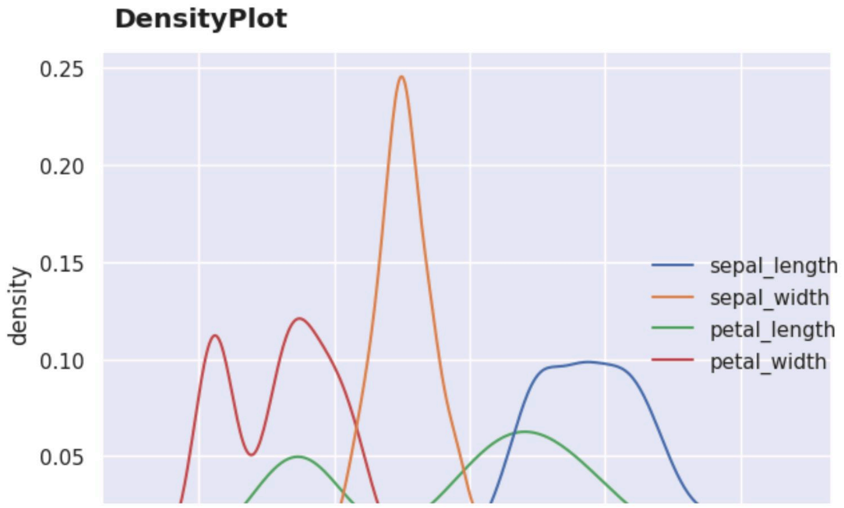

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/content/Iris.csv')
plt.figure(figsize=(10, 6))
plt.hist(df['PetalLengthCm'], bins=20, color='blue')
plt.title('Histogram')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
sns.histplot(df['SepalLengthCm'], label='Sepal Length')
plt.show()
sns.histplot(df['SepalWidthCm'], label='Sepal Width')
plt.show()
```


Histogram
y-axis
x-axis
0
5
10
15
20
25
30
1
2
3
4
5
6
7

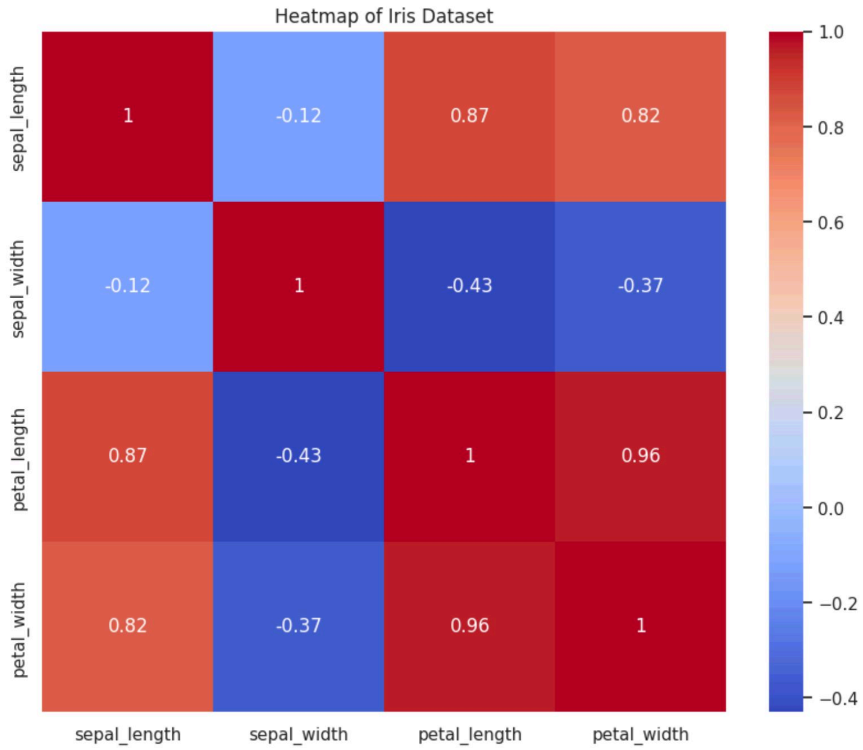

Count
SepalLengthCm
0
5
10
15
20
25
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Count
SepalWidthCm
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = sns.load_dataset('iris')

sns.displot(df, kind="kde", color = 'black')
plt.suptitle("DensityPlot", x=0.149, y=0.96, ha='left', fontweight = 'bold')
plt.xlabel("x-axis")
plt.ylabel("density")
plt.tight_layout()
plt.show()
```



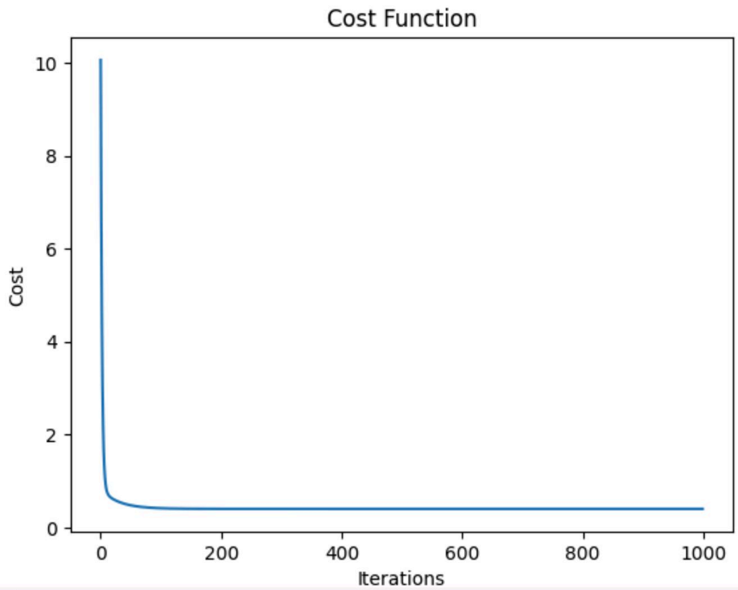

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = sns.load_dataset('iris')
df_numeric = df.select_dtypes(include=[int, float])
corr = df_numeric.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Heatmap of Iris Dataset')
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)


X_b = np.c_[np.ones((100, 1)), X]


def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1/(2*m)) * np.sum(np.square(predictions - y))
    return cost


def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = np.dot(X.transpose(), (predictions - y))
        theta -= (alpha/m) * errors
        cost_history[i] = compute_cost(X, y, theta)

    return theta, cost_history


theta = np.random.randn(2,1)

alpha = 0.1
iterations = 1000
theta, cost_history = gradient_descent(X_b, y, theta, alpha, iterations)

print("Theta:", theta)
print("Final cost:", cost_history[-1])
```

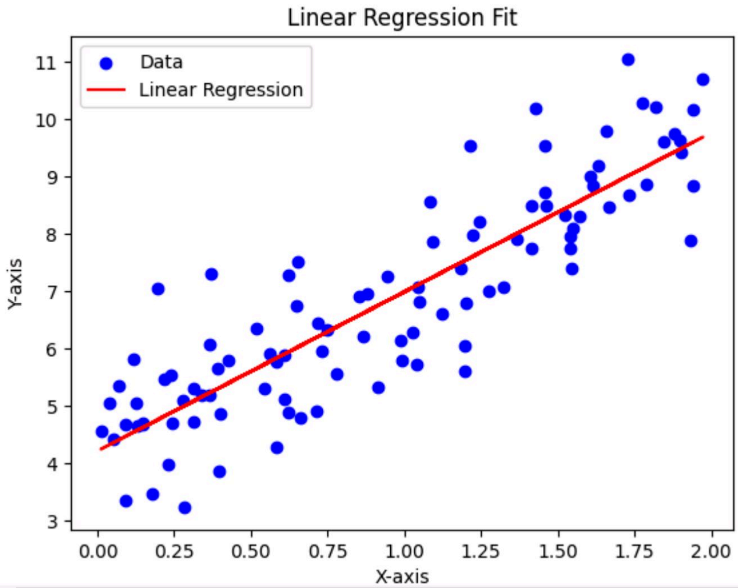
```
Theta: [[4.21509609]
 [2.77011344]]
Final cost: 0.4032922819835273
```

```
plt.plot(range(iterations), cost_history)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Function")
plt.show()
```




```
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, X_b.dot(theta), color='red', label='Linear Regression')
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Linear Regression Fit")
plt.legend()
plt.show()
```

```
plt.show()
```




```
import pandas as pd
ds= pd.read_csv("/content/housing.csv")
ds.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_va
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	45260
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	35850
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	35210
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	34130
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	34220

Next steps: Generate code with ds | View recommended plots | New interactive sheet

```
ds=ds.dropna()
```

```
X = ds.drop('median_house_value', axis=1)
y = ds['median_house_value']
```

```
X = pd.get_dummies(X, drop_first=True)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = model.predict(X_test)

from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R^2 Score:", r2)
```

```
Mean Squared Error: 4802173538.60416
R^2 Score: 0.6488402154431994
```

```
import pandas as pd
data=pd.read_csv("/content/HR.csv")
data.head()
```

	ID	Name	Department	GEO	Role	Rising_Star	Will_Relocate	Critical	Trending Perf	Talent_Level	...	salary	Gender
0	1	BRADDY	Operations	US	VP	3	0	1	8	6	...	low	M
1	2	BORST	Sales	UK	Senior Director	4	0	1	10	8	...	low	F
2	3	BIRDWELL	Finance	France	Senior Director	1	0	0	2	3	...	medium	F
3	4	BENT	Human Resources	China	Senior Director	4	0	1	8	7	...	high	M
4	5	BAZAN	IT	Korea	Director	4	0	1	8	7	...	low	F

5 rows × 30 columns

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = data.drop(['Department', 'GEO', 'Role', 'ID', 'Name'], axis=1)

salary_mapping = {'low': 0, 'medium': 1, 'high': 2}
data['salary'] = data['salary'].map(salary_mapping)

data = data.dropna()

X = data.select_dtypes(include=[float, int]).drop('salary', axis=1)
y = data['salary']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R^2 Score:", r2)
```

```
Mean Squared Error: 0.3840223282761343
R^2 Score: 0.14492463243664266
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df=pd.read_csv('/content/car_evaluation.csv')
```

```
df.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

```
df.shape
```

```
(1727, 7)
```

```
col_names = ['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety', 'class']

df.columns = col_names
col_names
```

```
['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1727 non-null   object
 1   meant     1727 non-null   object
 2   doors     1727 non-null   object
 3   persons   1727 non-null   object
 4   lug_boot  1727 non-null   object
 5   safety    1727 non-null   object
 6   class     1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
df['class'].value_counts()
```

	count
class	
unacc	1209
acc	384
good	69
vgood	65

dtype: int64

```
X = df.drop(['class'], axis=1)

y = df['class']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=42)

X_train.shape, X_test.shape
```

```
((1157, 6), (570, 6))
```

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.3.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.1.4)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encod
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2024
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (20
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoder
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoder
Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 2.0 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
# Encode Categorical
import category_encoders as ce

# encode variables with ordinal encoding
encoder = ce.OrdinalEncoder(cols=['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety'])

X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)

X_train.head()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-3-3e45ef93a379> in <cell line: 2>()
      1 # Encode Categorical
----> 2 import category_encoders as ce
      3 
      4 # encode variables with ordinal encoding
      5 encoder = ce.OrdinalEncoder(cols=['buying', 'meant', 'doors', 'persons', 'lug_boot', 'safety'])

ModuleNotFoundError: No module named 'category_encoders'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Next steps: Explain error

```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

```
y_pred_test = logreg.predict(X_test)
```

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.7702
```

```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.7891
```

```
# fit the Logsitic Regression model with C=100

# instantiate the model
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)


# fit the model
logreg100.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.7986
Test set score: 0.7754
```

```
from sklearn.model_selection import GridSearchCV


parameters = [{'C':[1, 10, 100, 1000]}]



grid_search = GridSearchCV(estimator = logreg,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)


grid_search.fit(X_train, y_train)
```

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
# examine the best model

# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.7952


Parameters that give the best results :

 {'C': 1000}


Estimator that was chosen by the search :

 LogisticRegression(C=1000, random_state=0, solver='liblinear')
```

```
# calculate GridSearch CV score on test set
```

```
print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
GridSearch CV score on test set: 0.7754
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

distributions = dict(C=uniform(loc=0, scale=4),
                     penalty=['l2', 'l1'])

randomized_search = RandomizedSearchCV(estimator = logreg,
                                param_distributions = distributions,
                                scoring = 'accuracy',
                                cv = 5,
                                verbose=0)
```

```
randomized_search.fit(X_train, y_train)
```

```
▸      RandomizedSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

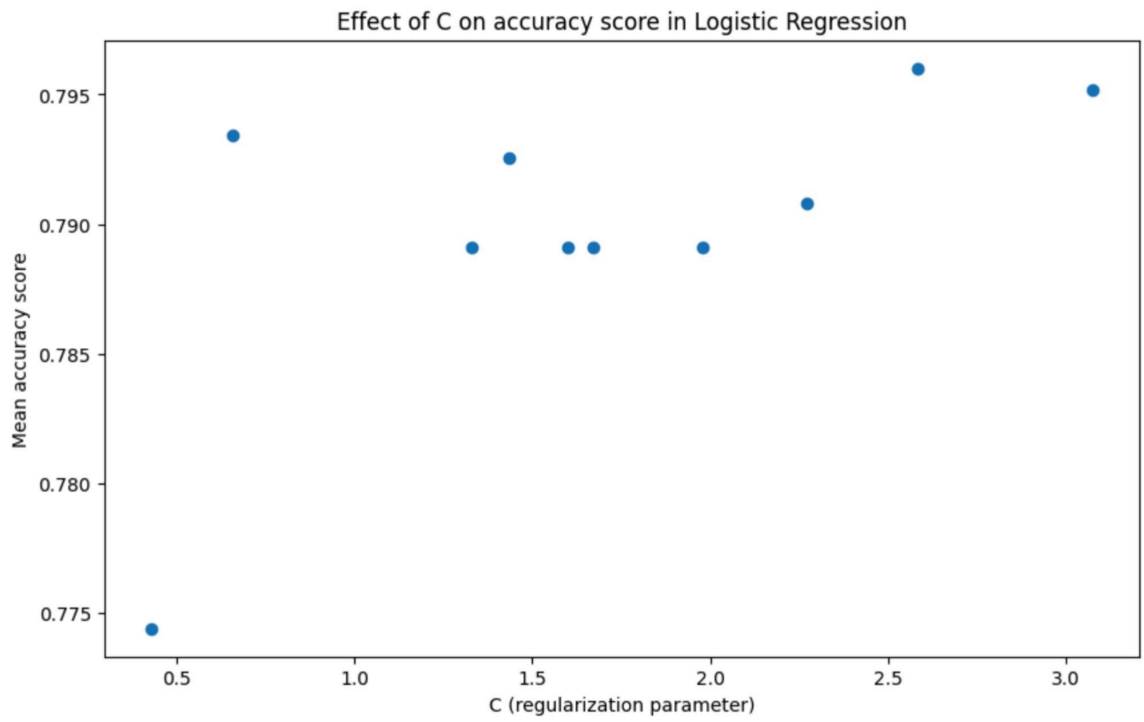
```
# examine the best model

# best score achieved during the GridSearchCV
print('RandomizedSearch CV best score : {:.4f}\n\n'.format(randomized_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (randomized_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (randomized_search.best_estimator_))
```

```
RandomizedSearch CV best score : 0.7960


Parameters that give the best results :

 {'C': 2.581310528951185, 'penalty': 'l1'}


Estimator that was chosen by the search :

 LogisticRegression(C=2.581310528951185, penalty='l1', random_state=0,
                   solver='liblinear')
```

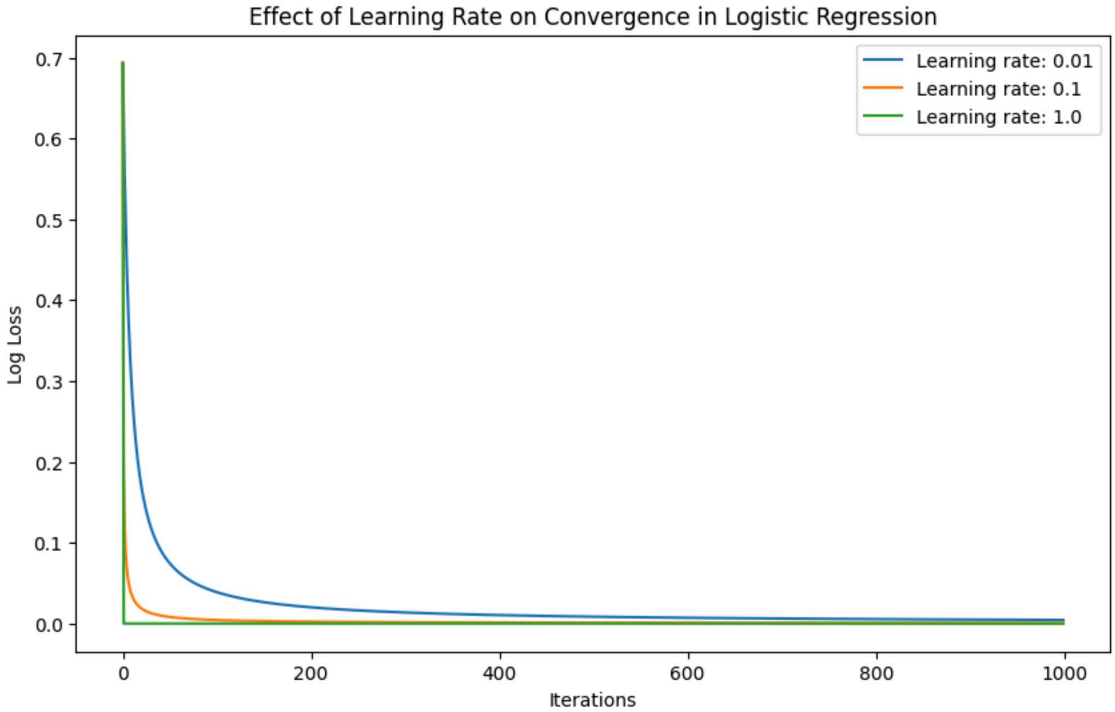
```
# calculate RandomizedSearch CV score on test set

print(' score on test set: {0:0.4f}'.format(randomized_search.score(X_test, y_test)))
```

```
 score on test set: 0.7719
```

```
# Plot the results
results = pd.DataFrame(randomized_search.cv_results_)
plt.figure(figsize=(10, 6))
plt.scatter(results['param_C'], results['mean_test_score'])
plt.xlabel('C (regularization parameter)')
plt.ylabel('Mean accuracy score')
plt.title('Effect of C on accuracy score in Logistic Regression')
plt.show()
```

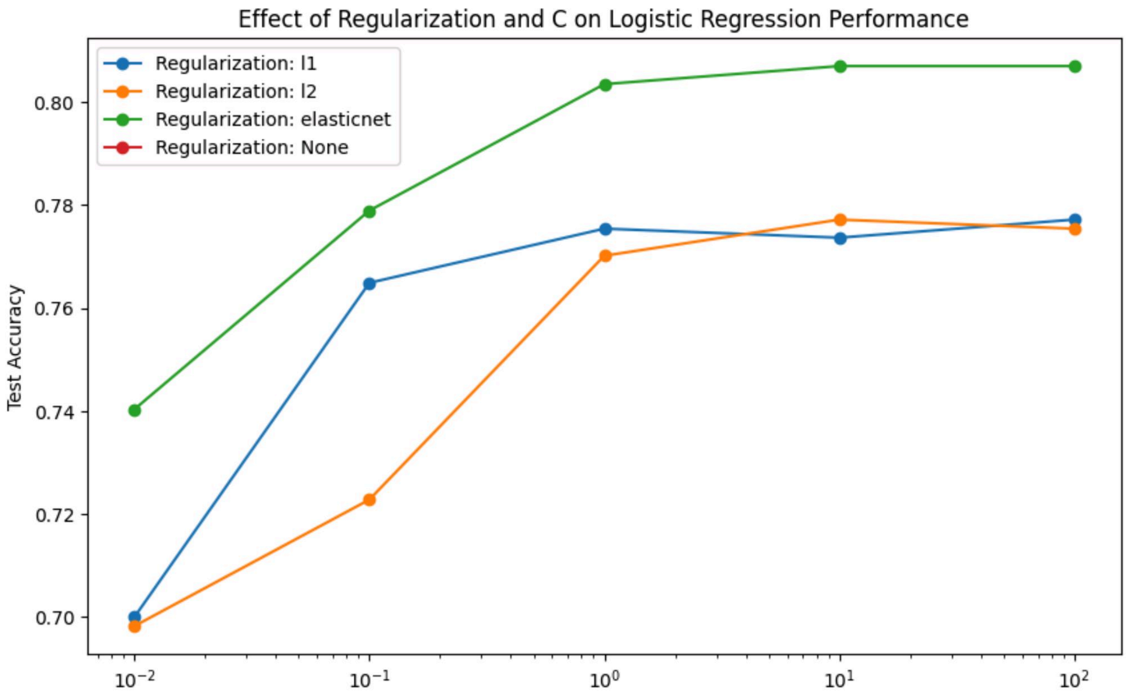



```
#task2
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def log_loss(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

def logistic_regression_gd(X, y, learning_rate, num_iterations):
    m, n = X.shape
    theta = np.zeros(n)
    loss_history = []

    for _ in range(num_iterations):
        z = np.dot(X, theta)
        h = sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / m
        theta -= learning_rate * gradient
        loss_history.append(log_loss(y, h))

    return theta, loss_history

# Prepare data for binary classification (setosa vs. not setosa)
y_binary = (y_train == 0).astype(int)
X_train_with_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]

learning_rates = [0.01, 0.1, 1.0]
num_iterations = 1000

plt.figure(figsize=(10, 6))
for lr in learning_rates:
    theta, loss_history = logistic_regression_gd(X_train_with_bias, y_binary, lr, num_iterations)
    plt.plot(range(num_iterations), loss_history, label=f'Learning rate: {lr}')

plt.xlabel('Iterations')
plt.ylabel('Log Loss')
plt.title('Effect of Learning Rate on Convergence in Logistic Regression')
plt.legend()
plt.show()
```




```
#task3
regularizations = ['l1', 'l2', 'elasticnet', None]
C_values = [0.01, 0.1, 1, 10, 100]

results = []

for reg in regularizations:
    for C in C_values:
        if reg == 'elasticnet':
            model = LogisticRegression(penalty=reg, solver='saga', C=C, l1_ratio=0.5, random_state=42, max_iter=500)
        elif reg is None:
            model = LogisticRegression(penalty=reg, solver='lbfgs', C=C, random_state=42, max_iter=500)
        else:
            model = LogisticRegression(penalty=reg, solver='liblinear', C=C, random_state=42, max_iter=500)

        model.fit(X_train, y_train)
        train_score = model.score(X_train, y_train)
        test_score = model.score(X_test, y_test)
        results.append((reg, C, train_score, test_score))

results_df = pd.DataFrame(results, columns=['Regularization', 'C', 'Train Score', 'Test Score'])

plt.figure(figsize=(10, 6))
for reg in regularizations:
    reg_results = results_df[results_df['Regularization'] == reg]
    plt.plot(reg_results['C'], reg_results['Test Score'], marker='o', label=f'Regularization: {reg}')

plt.xscale('log')
plt.xlabel('C (inverse of regularization strength)')
plt.ylabel('Test Accuracy')
plt.title('Effect of Regularization and C on Logistic Regression Performance')
plt.legend()
plt.show()

print(results_df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1193: UserWarning: Setting penalty=None will ignore the C
  warnings.warn(
```



```
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
%matplotlib inline
df=pd.read_csv('/content/car_evaluation.csv')
df.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(1727, 7)
```

```
col_names= ['buying','maint', 'doors', 'persons', 'lug_boot', 'safety' ,'class']
df.columns=col_names
col_names
```

```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1727 non-null   object
 1   maint     1727 non-null   object
 2   doors     1727 non-null   object
 3   persons   1727 non-null   object
 4   lug_boot  1727 non-null   object
 5   safety    1727 non-null   object
 6   class     1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
df['class'].value_counts()
```

class	count
unacc	1209
acc	384
good	69
vgood	65

```
X = df.drop(['class'] , axis = 1)
Y = df['class']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test,Y_train,Y_test = train_test_split(X , Y ,test_size = 0.33 , random_state = 42)

X_train.shape,X_test.shape
```

```
((1157, 6), (570, 6))
```

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.26.4)
```

```
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.3.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.1.4)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_enco
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (202
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encode
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encode
Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 3.4 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
import category_encoders as ce
encoder=ce.OrdinalEncoder(cols=['buying','maint', 'doors', 'persons', 'lug_boot', 'safety' ])

X_train=encoder.fit_transform(X_train)
X_test=encoder.transform(X_test)
X_train.head()
```

	buying	maint	doors	persons	lug_boot	safety
83	1	1	1	1	1	1
48	1	1	2	2	1	2
468	2	1	2	3	2	2
155	1	2	2	2	1	1
1043	3	2	3	2	2	1

Next steps: Generate code with X_train View recommended plots New interactive sheet

```
from sklearn.tree import DecisionTreeClassifier
clf_gini = DecisionTreeClassifier(criterion='gini',max_depth=3,random_state=0)
clf_gini.fit(X_train,Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
Y_pred_gini=clf_gini.predict(X_test)
Y_pred_gini[:5]
```

```
array(['unacc', 'unacc', 'unacc', 'acc', 'unacc'], dtype=object)
```

```
from sklearn.metrics import accuracy_score
print("Model Accuracy score with prediction for test dataset with gini index {0:0.4f}".format(accuracy_score(Y_pred_gini,Y_test)))
```

```
Model Accuracy score with prediction for test dataset with gini index 0.8053
```

```
Y_pred_train_gini=clf_gini.predict(X_train)
Y_pred_train_gini
```

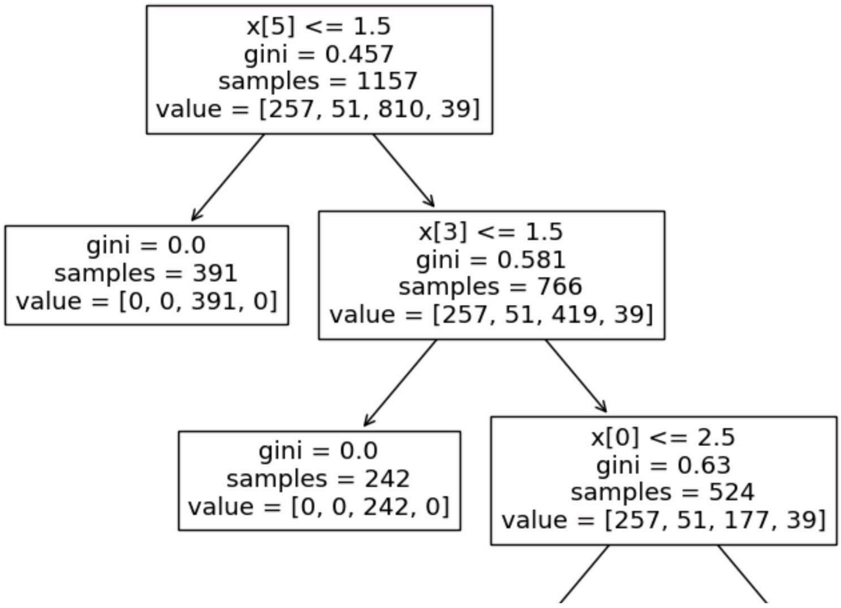
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

```
print("Model Accuracy score with prediction for training dataset with gini index {0:0.4f}".format(accuracy_score(Y_pred_train_gini,Y_tr
```

```
Model Accuracy score with prediction for training dataset with gini index 0.7848
```

```
import matplotlib.pyplot as plt
from sklearn import tree
plt.figure(figsize=(10,8))
tree.plot_tree(clf_gini.fit(X_train,Y_train))
```

```
[Text(0.3333333333333333, 0.875, 'x[5] <= 1.5\ngini = 0.457\nsamples = 1157\nvalue = [257, 51, 810, 39]'),
 Text(0.16666666666666666, 0.625, 'gini = 0.0\nsamples = 391\nvalue = [0, 0, 391, 0]'),
 Text(0.5, 0.625, 'x[3] <= 1.5\ngini = 0.581\nsamples = 766\nvalue = [257, 51, 419, 39]'),
 Text(0.3333333333333333, 0.375, 'gini = 0.0\nsamples = 242\nvalue = [0, 0, 242, 0]'),
 Text(0.6666666666666666, 0.375, 'x[0] <= 2.5\ngini = 0.63\nsamples = 524\nvalue = [257, 51, 177, 39]'),
 Text(0.5, 0.125, 'gini = 0.498\nsamples = 266\nvalue = [124, 0, 142, 0]'),
 Text(0.8333333333333334, 0.125, 'gini = 0.654\nsamples = 258\nvalue = [133, 51, 35, 39]')]
```




```
#Using Gaussian Naive Bias
#The Naive Bayes classifier is a probabilistic model based on Bayes’
#theorem which is used to calculate the probability P(A|B) of an event A occurring, when we are given some prior knowledge B
```



```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB(priors=[0.6, 0.3, 0.1, 0.0])

gnb.fit(X_train, Y_train)

print("print Train for accuracy of NBC algo: ", gnb.score(X_train,Y_train))
print("print Test for accuracy of NBC algo: ", gnb.score(X_test,Y_test))
```

```
print Train for accuracy of NBC algo:  0.7519446845289542
print Test for accuracy of NBC algo:  0.7403508771929824
/usr/local/lib/python3.10/dist-packages/sklearn/naive_bayes.py:509: RuntimeWarning: divide by zero encountered in log
  jointi = np.log(self.class_prior_[i])
/usr/local/lib/python3.10/dist-packages/sklearn/naive_bayes.py:509: RuntimeWarning: divide by zero encountered in log
  jointi = np.log(self.class_prior_[i])
```

```
#In summation, Naive Bayes’ independence assumption is a crucial factor for the classifier’s success.
#We have to make sure it applies (to some degree) to our data before we can properly utilize it.
#Likewise, Decision Trees are dependent on proper pruning techniques so that overfitting can be avoided while
#keeping track of the classification objective.
#All in all, they are both very useful methods and a great addition to our toolkit.
```

```
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import warnings

warnings.filterwarnings('ignore')
```

```
data = '/content/car_evaluation.csv'

df = pd.read_csv(data, header=None)
```

```
# view dimensions of dataset

df.shape
```

```
(1728, 7)
```

```
# preview the dataset

df.head()
```

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Next steps: Generate code with df View recommended plots New interactive sheet

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
df.columns = col_names

col_names
```

```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
# let's again preview the dataset

df.head()
```

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   object
```

```
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']


for col in col_names:

    print(df[col].value_counts())
```

```
buying
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
maint
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
doors
2        432
3        432
4        432
5more    432
Name: count, dtype: int64
persons
2       576
4       576
more    576
Name: count, dtype: int64
lug_boot
small    576
med      576
big      576
Name: count, dtype: int64
safety
low     576
med     576
high    576
Name: count, dtype: int64
class
unacc    1210
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

```
df['class'].value_counts()
```

	count
class	
unacc	1210
acc	384
good	69
vgood	65

```
# check missing values in variables

df.isnull().sum()
```

	0
buying	0
maint	0
doors	0
persons	0
lug_boot	0
safety	0
class	0

```
X = df.drop(['class'], axis=1)

y = df['class']


# split data into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)


# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

```
((1157, 6), (571, 6))
```

```
# check data types in X_train

X_train.dtypes
```

	0
buying	object
maint	object
doors	object
persons	object
lug_boot	object
safety	object

```
X_train.head()
```

	buying	maint	doors	persons	lug_boot	safety
48	vhigh	vhigh	3	more	med	low
468	high	vhigh	3	4	small	low
155	vhigh	high	3	more	small	high
1721	low	low	5more	more	small	high
1208	med	low	2	more	small	high

Next steps: Generate code with X_train View recommended plots New interactive sheet

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.3.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.1.4)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_enco
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (202
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encode
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encode
Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 2.5 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
# import category encoders

import category_encoders as ce


# encode categorical variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)


X_train.head()
```

	buying	maint	doors	persons	lug_boot	safety
48	1	1	1	1	1	1
468	2	1	1	2	2	1
155	1	2	1	1	2	2
1721	3	3	2	1	2	2
1208	4	3	3	1	2	2

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

```
# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

# instantiate the classifier

rfc = RandomForestClassifier(random_state=0)

# fit the model

rfc.fit(X_train, y_train)

# Predict the Test set results

y_pred = rfc.predict(X_test)



# Check accuracy score

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with 10 decision-trees : 0.9457
```

```
# instantiate the classifier with n_estimators = 100

rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)

# fit the model to the training set

rfc_100.fit(X_train, y_train)

# Predict on the test set results

y_pred_100 = rfc_100.predict(X_test)



# Check accuracy score
```

```
print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))
```

```
Model accuracy score with 100 decision-trees : 0.9457
```

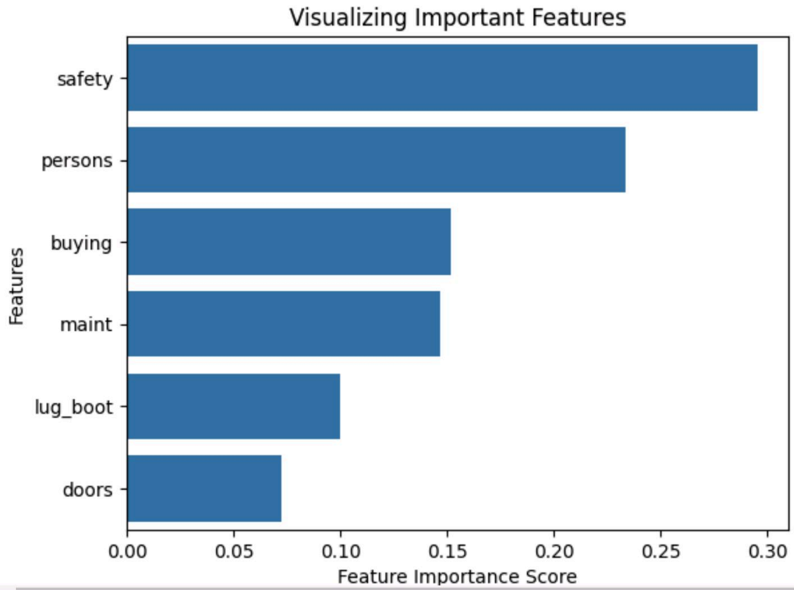
```
# create the classifier with n_estimators = 100

clf = RandomForestClassifier(n_estimators=100, random_state=0)



# fit the model to the training set

clf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
feature_scores = pd.Series(clf.feature_importances_, index=X_train.columns).sort_values(ascending=False)

feature_scores
```

	0
safety	0.295319
persons	0.233856
buying	0.151734
maint	0.146653
lug_boot	0.100048
doors	0.072389

```
# Creating a seaborn bar plot

sns.barplot(x=feature_scores, y=feature_scores.index)

# Add labels to the graph

plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

# Add title to the graph

plt.title("Visualizing Important Features")

# Visualize the graph

plt.show()
```



```
# declare feature vector and target variable

X = df.drop(['class', 'doors'], axis=1)

y = df['class']

# split data into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)

# encode categorical variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'persons', 'lug_boot', 'safety'])


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)

# instantiate the classifier with n_estimators = 100

clf = RandomForestClassifier(random_state=0)



# fit the model to the training set

clf.fit(X_train, y_train)

# Predict on the test set results

y_pred = clf.predict(X_test)



# Check accuracy score

print('Model accuracy score with doors variable removed : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with doors variable removed : 0.9264
```

Start coding or generate with AI.

```
# DataFlair Iris Flower Classification
# Import Packages
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
%matplotlib inline
```

```
 columns = ['Sepal length', 'Sepal width', 'Petal length', 'Petal width', 'Class_labels']
# Load the data
df = pd.read_csv('iris.data', names=columns)
df.head()
```

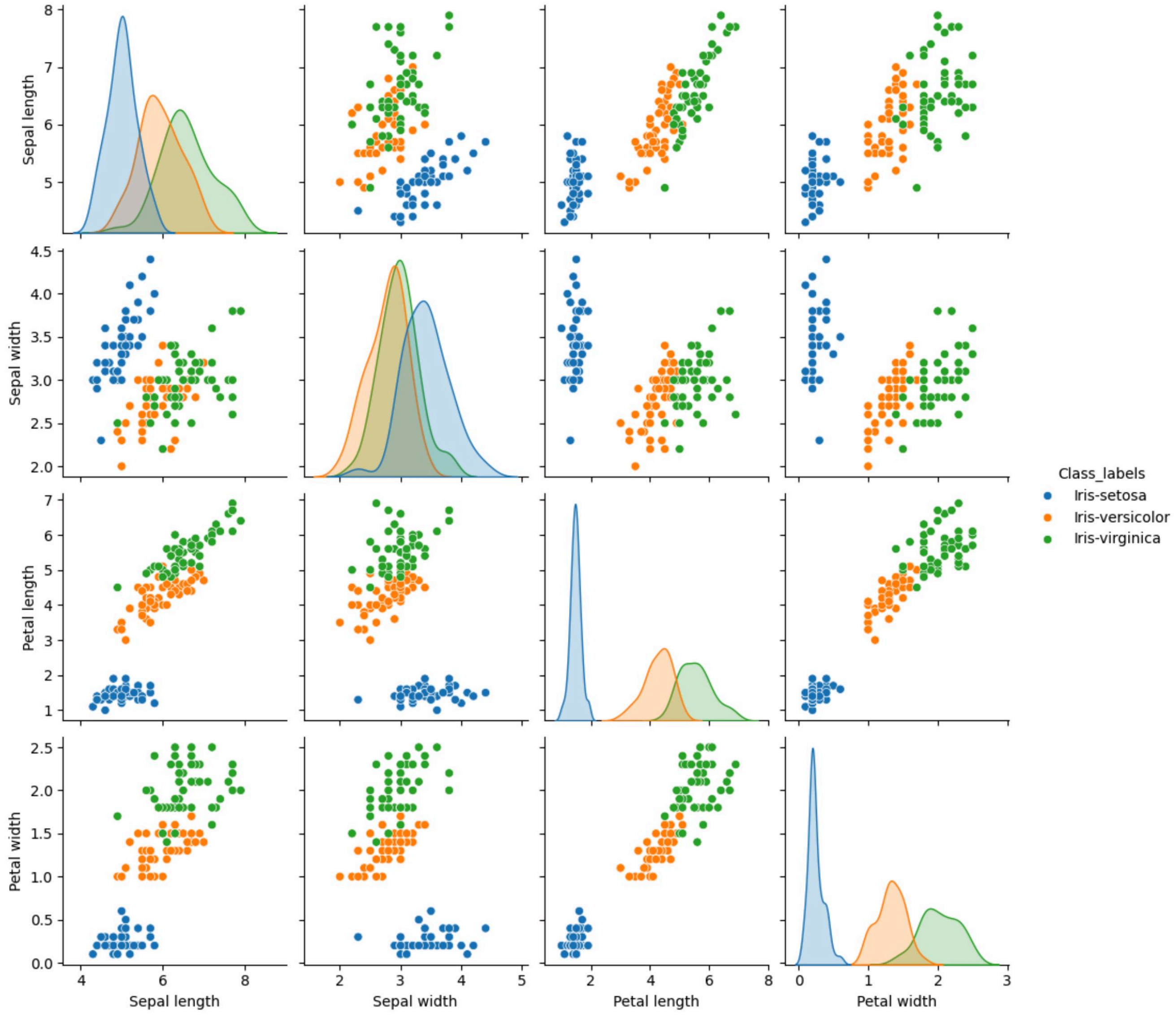
	Sepal length	Sepal width	Petal length	Petal width	Class_labels
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# Some basic statistical analysis about the data
df.describe()
```

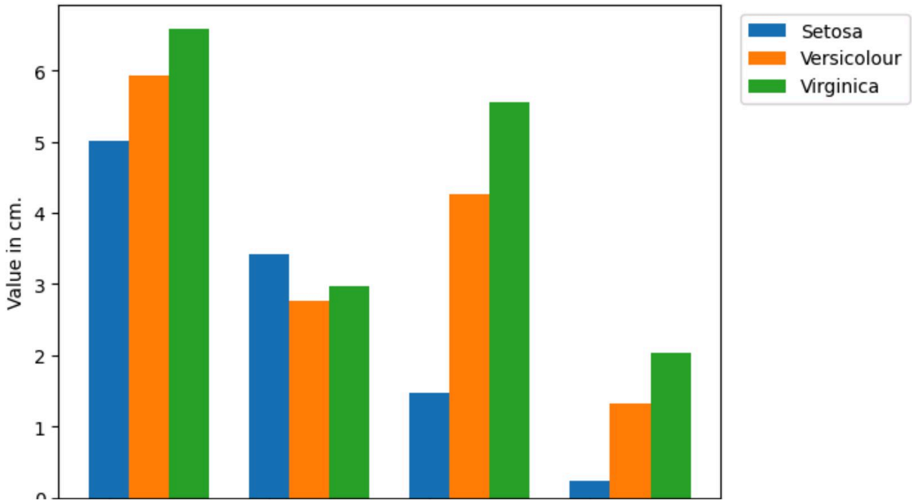
	Sepal length	Sepal width	Petal length	Petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
# Visualize the whole dataset
sns.pairplot(df, hue='Class_labels')
```

```
<seaborn.axisgrid.PairGrid at 0x7d19b001ef80>
```




```
# Separate features and target
data = df.values
X = data[:,0:4]
Y = data[:,4]


# Calculate average of each features for all classes
Y_Data = np.array([np.average(X[:, i][Y==j].astype('float32')) for i in range (X.shape[1])
 for j in (np.unique(Y))])
Y_Data_reshaped = Y_Data.reshape(4, 3)
Y_Data_reshaped = np.swapaxes(Y_Data_reshaped, 0, 1)
X_axis = np.arange(len(columns)-1)
width = 0.25


# Plot the average
plt.bar(X_axis, Y_Data_reshaped[0], width, label = 'Setosa')
plt.bar(X_axis+width, Y_Data_reshaped[1], width, label = 'Versicolour')
plt.bar(X_axis+width*2, Y_Data_reshaped[2], width, label = 'Virginica')
plt.xticks(X_axis, columns[:4])
plt.xlabel("Features")
plt.ylabel("Value in cm.")
plt.legend(bbox_to_anchor=(1.3,1))
plt.show()
```




```
# Split the data to train and test dataset.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)


# Support vector machine algorithm
from sklearn.svm import SVC
svn = SVC()
svn.fit(X_train, y_train)
```

```
SVC
SVC()
```

```
# Predict from the test dataset
predictions = svn.predict(X_test)

# Calculate the accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
```

```
0.9666666666666667
```

Start coding or generate with AI.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import Normalizer
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```
df = pd.read_csv('/content/Country-data.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   country     167 non-null    object
 1   child_mort  167 non-null    float64
 2   exports     167 non-null    float64
 3   health      167 non-null    float64
 4   imports     167 non-null    float64
 5   income      167 non-null    int64
 6   inflation   167 non-null    float64
 7   life_expec  167 non-null    float64
 8   total_fer   167 non-null    float64
 9   gdpp        167 non-null    int64
dtypes: float64(7), int64(2), object(1)
memory usage: 13.2+ KB
```

```
dataframe = df.copy()
dataframe.drop(columns=['country'], inplace=True)
dataframe
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200
...	...	...	...	...	...	...	...	...	...
162	29.2	46.6	5.25	52.7	2950	2.62	63.0	3.50	2970
163	17.1	28.5	4.91	17.6	16500	45.90	75.4	2.47	13500
164	23.3	72.0	6.84	80.2	4490	12.10	73.1	1.95	1310
165	56.3	30.0	5.18	34.4	4480	23.60	67.5	4.67	1310
166	83.1	37.0	5.89	30.9	3280	14.00	52.0	5.40	1460

167 rows × 9 columns

```
values = Normalizer().fit_transform(dataframe.values)
print(values)
```

```
[[5.28625544e-02 5.86059362e-03 4.44232996e-03 ... 3.29365361e-02
  3.41086549e-03 3.24090827e-01]
 [1.54565929e-03 2.60713615e-03 6.09883634e-04 ... 7.10444600e-03
  1.53634809e-04 3.80828101e-01]
 [2.00006203e-03 2.81327406e-03 3.05503980e-04 ... 5.60456942e-03
  2.11728178e-04 3.26750061e-01]
 ...
 [4.97959888e-03 1.53876017e-02 1.46182216e-03 ... 1.56226900e-02
  4.16747546e-04 2.79968864e-01]
 [1.20589885e-02 6.42574875e-03 1.10951262e-03 ... 1.44579347e-02
  1.00027489e-03 2.80591029e-01]
 [2.31349866e-02 1.03007762e-02 1.63977221e-03 ... 1.44767666e-02
  1.50335653e-03 4.06463062e-01]]
```

```
def clustering_algorithm(n_clusters, dataset):
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, max_iter=300)
    labels = kmeans.fit_predict(dataset)
    s = metrics.silhouette_score(dataset, labels, metric='euclidean')
    dbs = metrics.davies_bouldin_score(dataset, labels)
    calinski = metrics.calinski_harabasz_score(dataset, labels)
    return s, dbs, calinski
```

```
for i in range(3, 11):
    s, dbs, calinski = clustering_algorithm(i, values)
    print(i, s, dbs, calinski)
```

```
3 0.5198837827909313 0.6008669317607285 458.31264276466067
4 0.4634990957986046 0.7218455631804814 439.8019905572253
5 0.43859967605023265 0.7710112898249146 417.24674177320094
6 0.4696101045315829 0.7520661493821988 442.4034615165842
7 0.4645904730917045 0.6773860500589699 457.84977996199217
8 0.425997367740665 0.7319353703488833 465.1721966231241
9 0.43663653633042926 0.7353285280488965 468.87261942843736
10 0.43579828501773965 0.6633562896255003 492.7731886302295
```

```
random_data = np.random.rand(167,9)
s_random, dbs_random, calinski_random = clustering_algorithm(3, random_data)
s, dbs, calinski = clustering_algorithm(3, values)

print(s_random, dbs_random, calinski_random)
print(s, dbs, calinski)
```

```
0.09288154412420664 2.518987701787166 17.920394992597448
0.5198837827909313 0.6008669317607285 458.3126427646605
```

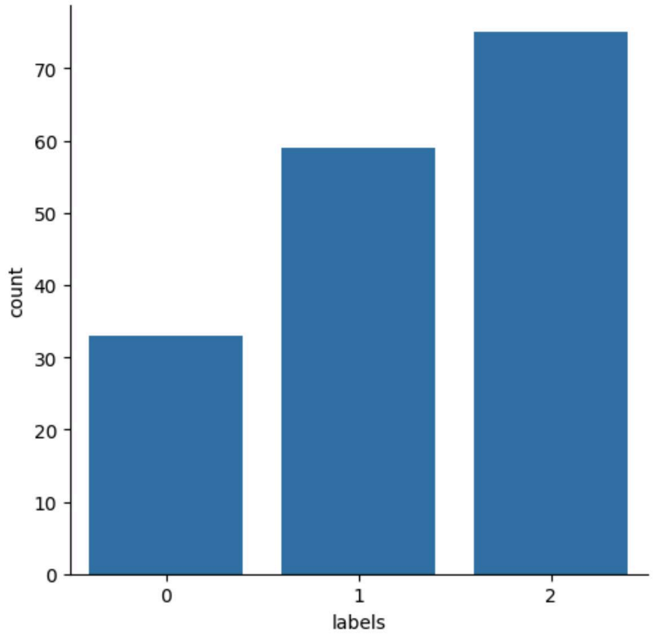
```
set1, set2, set3 = np.array_split(values, 3)
s1, dbs1, calinski1 = clustering_algorithm(3, set1)
s2, dbs2, calinski2 = clustering_algorithm(3, set2)
s3, dbs3, calinski3 = clustering_algorithm(3, set3)
print(s1, dbs1, calinski1)
print(s2, dbs2, calinski2)
print(s3, dbs3, calinski3)
```

```
0.5099002406186719 0.617677169564452 163.35312834956085
0.5355580436538645 0.5880283946904726 188.30786716669212
0.5657004742226224 0.5356671502718732 142.91946826594048
```

```
kmeans = KMeans(n_clusters=3, n_init=10, max_iter=300)
y_pred = kmeans.fit_predict(values)
labels = kmeans.labels_

df['labels'] = labels
```

```
sns.catplot(x='labels', kind='count', data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7e367bb27760>
```




```
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[1.89973424e-03 1.41159823e-03 5.18643396e-04 2.90914866e-03
  6.78742436e-01 3.45565877e-04 3.63893463e-03 1.78818699e-04
  7.29344202e-01]
 [1.07197437e-02 5.20213097e-03 9.02207865e-04 7.28342214e-03
  8.63230309e-01 1.23833105e-03 9.06304745e-03 6.08413040e-04
```

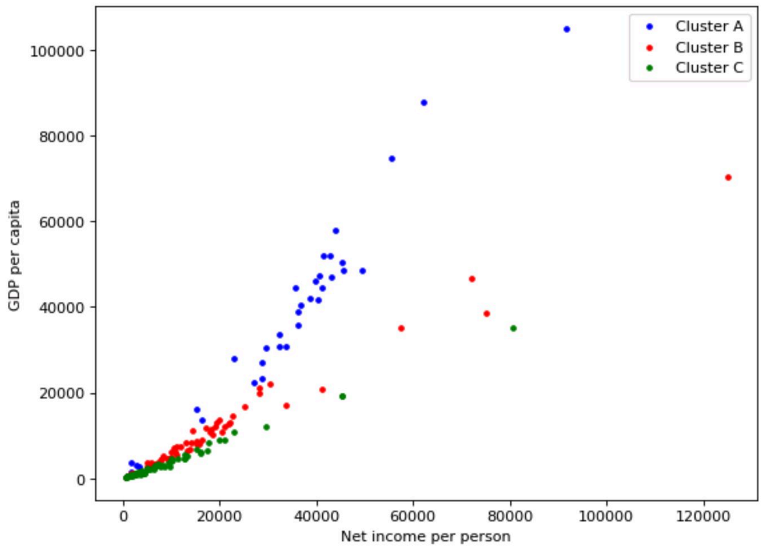
```
  4.99951321e-01]
 [2.64610171e-02 8.03462903e-03 2.14032264e-03 1.39988062e-02
  9.31344877e-01 2.92813714e-03 1.95943709e-02 1.48105369e-03
  3.56104512e-01]]
```

```
max = len(centroids[0])
for i in range(max):
    print(dataframe.columns.values[i],"\n{:.4f}".format(centroids[:, i].var()))
```

```
child_mort
0.0001
exports
0.0000
health
0.0000
imports
0.0000
income
0.0114
inflation
0.0000
life_expec
0.0000
total_fer
0.0000
gdpp
0.0236
```

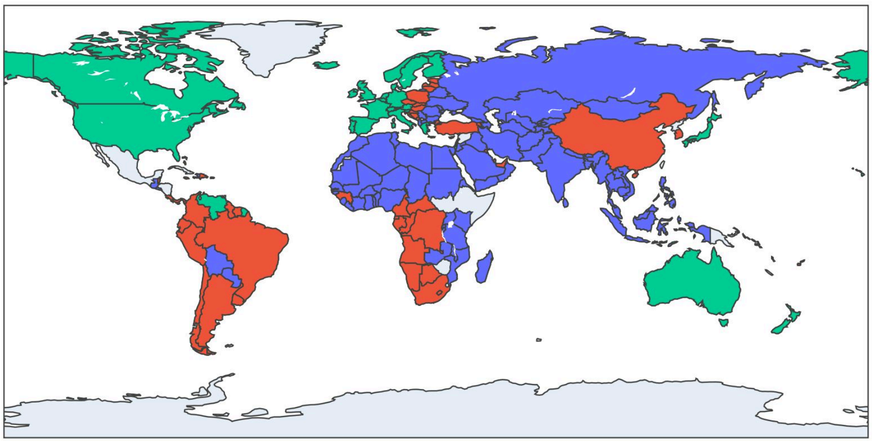
```
df_0 = df[df['labels'] == 0]
df_1 = df[df['labels'] == 1]
df_2 = df[df['labels'] == 2]
```

```
plt.figure(figsize=(8, 6), dpi=80)
plt.scatter(df_0['income'], df_0['gdpp'], c='blue', s=10, label='Cluster A')
plt.scatter(df_1['income'], df_1['gdpp'], c='red', s=10, label='Cluster B')
plt.scatter(df_2['income'], df_2['gdpp'], c='green', s=10, label='Cluster C')

plt.xlabel('Net income per person')
plt.ylabel('GDP per capita')
plt.legend(),
plt.show()
```




```
clusters_name = {0: 'Cluster A', 1: 'Cluster B', 2: 'Cluster C'}
df['labels'] = df['labels'].map(clusters_name)

fig = px.choropleth(df,
                    locationmode='country names',
                    locations='country',
                    color='labels',
                    title='Coutries by labels'
                   )
fig.show()
```

Coutries by labels

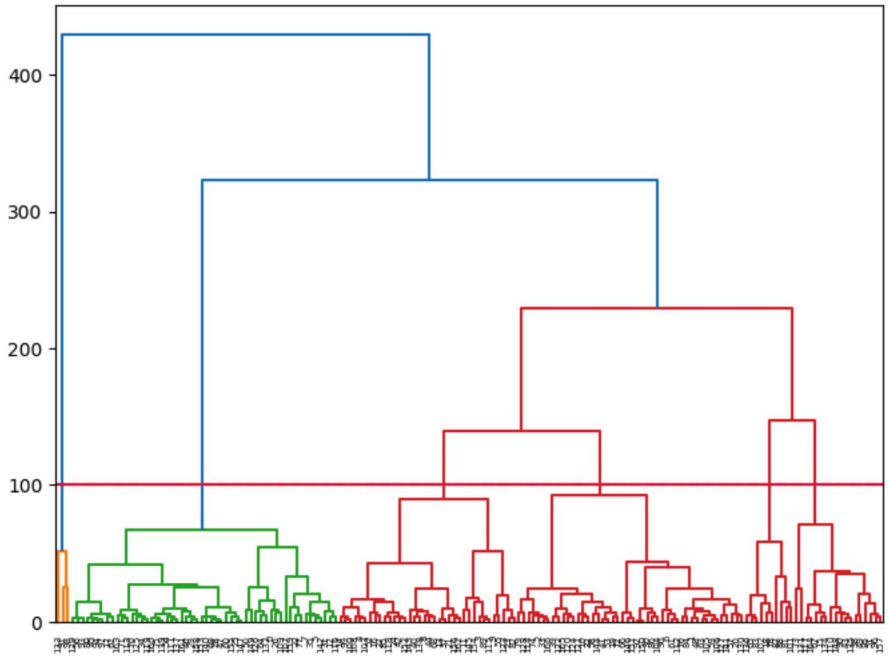


```
description = df.groupby("labels")[['child_mort', 'exports', 'health', 'imports', 'income', 'inflation', 'life_expec', 'total_fer', 'gdp
n_clients = description.size()
description = description.mean()
description['n_clients'] = n_clients
print(description)
```

```
           child_mort    exports    health    imports        income  \
labels
Cluster A   10.545455  42.875758  9.866667  45.257576  34696.060606
Cluster B   31.310169  49.176271  6.318644  53.616949  18212.322034
Cluster C   55.944000  33.985320  5.864267  42.316879   8582.213333

           inflation  life_expec  total_fer          gdpp  n_clients
labels
Cluster A   3.503455   78.403030   2.103333  38552.121212         33
Cluster B   5.953746   71.157627   2.650678  10734.186441         59
Cluster C  11.102413   66.629333   3.553467   3459.693333         75
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster import hierarchy

df = pd.read_csv('/content/Country-data.csv')
print(df.shape)
df = df[['imports','exports', 'health']]
df = df.dropna(axis=0)
clusters = hierarchy.linkage(df, method="ward")

plt.figure(figsize=(8, 6))
dendrogram = hierarchy.dendrogram(clusters)
plt.axhline(100, color='red', linestyle='--');
plt.axhline(100, color='crimson');
```

```
(167, 10)
```



```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.31.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.12.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import time
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('/content/Iris.csv')

df = df.drop(columns=['Id'], errors='ignore')

encoder = LabelEncoder()
df['Species'] = encoder.fit_transform(df['Species'])

X = df.drop('Species', axis=1).values
y = to_categorical(df['Species'].values)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
df_setosa = df[df['Species'] == 0].head(10)
df_versicolor = df[df['Species'] == 1].head(10)
df_virginica = df[df['Species'] == 2].head(10)
print(df_setosa)
print(df_versicolor)
print(df_virginica)
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0            5.1           3.5            1.4           0.2        0
1            4.9           3.0            1.4           0.2        0
```

```
2            4.7           3.2            1.3           0.2        0
3            4.6           3.1            1.5           0.2        0
4            5.0           3.6            1.4           0.2        0
5            5.4           3.9            1.7           0.4        0
6            4.6           3.4            1.4           0.3        0
7            5.0           3.4            1.5           0.2        0
8            4.4           2.9            1.4           0.2        0
9            4.9           3.1            1.5           0.1        0
    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
50            7.0           3.2            4.7           1.4        1
51            6.4           3.2            4.5           1.5        1
52            6.9           3.1            4.9           1.5        1
53            5.5           2.3            4.0           1.3        1
54            6.5           2.8            4.6           1.5        1
55            5.7           2.8            4.5           1.3        1
56            6.3           3.3            4.7           1.6        1
57            4.9           2.4            3.3           1.0        1
58            6.6           2.9            4.6           1.3        1
59            5.2           2.7            3.9           1.4        1
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
100            6.3           3.3            6.0           2.5        2
101            5.8           2.7            5.1           1.9        2
102            7.1           3.0            5.9           2.1        2
103            6.3           2.9            5.6           1.8        2
104            6.5           3.0            5.8           2.2        2
105            7.6           3.0            6.6           2.1        2
106            4.9           2.5            4.5           1.7        2
107            7.3           2.9            6.3           1.8        2
108            6.7           2.5            5.8           1.8        2
109            7.2           3.6            6.1           2.5        2
```

```python
model = Sequential([
    Dense(8, input_shape=(X_train.shape[1],), activation='relu'),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```python
model_improved = Sequential([
    Dense(8, input_shape=(X_train.shape[1],), activation='relu'),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])

model_improved.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])

history_improved = model_improved.fit(X_train, y_train, epochs=20, batch_size=10, validation_split=0.1, verbose=1)

loss_improved, accuracy_improved = model_improved.evaluate(X_test, y_test, verbose=1)
accuracy_improved *= 100
print(f'Accuracy: {accuracy_improved:.4f}')
```

```
Epoch 1/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.3661 - loss: 1.1538 - val_accuracy: 0.4167 - val_loss: 1.1244
Epoch 2/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.3617 - loss: 1.1097 - val_accuracy: 0.4167 - val_loss: 1.1012
Epoch 3/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3461 - loss: 1.0973 - val_accuracy: 0.4167 - val_loss: 1.0767
Epoch 4/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.3232 - loss: 1.0720 - val_accuracy: 0.4167 - val_loss: 1.0518
Epoch 5/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.3064 - loss: 1.0470 - val_accuracy: 0.4167 - val_loss: 1.0236
Epoch 6/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.2532 - loss: 1.0188 - val_accuracy: 0.4167 - val_loss: 0.9973
Epoch 7/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4052 - loss: 0.9783 - val_accuracy: 0.6667 - val_loss: 0.9693
Epoch 8/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3276 - loss: 0.9764 - val_accuracy: 0.8333 - val_loss: 0.9444
Epoch 9/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6146 - loss: 0.9219 - val_accuracy: 0.9167 - val_loss: 0.9185
Epoch 10/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7926 - loss: 0.9077 - val_accuracy: 0.9167 - val_loss: 0.8931
Epoch 11/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8262 - loss: 0.8750 - val_accuracy: 0.9167 - val_loss: 0.8712
Epoch 12/20
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7989 - loss: 0.8506 - val_accuracy: 0.9167 - val_loss: 0.8480
Epoch 13/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8025 - loss: 0.8322 - val_accuracy: 0.8333 - val_loss: 0.8237
Epoch 14/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7644 - loss: 0.7845 - val_accuracy: 0.8333 - val_loss: 0.7995
Epoch 15/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7639 - loss: 0.7425 - val_accuracy: 0.8333 - val_loss: 0.7754
Epoch 16/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7718 - loss: 0.7412 - val_accuracy: 0.8333 - val_loss: 0.7525
Epoch 17/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7367 - loss: 0.7144 - val_accuracy: 0.8333 - val_loss: 0.7295
Epoch 18/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7376 - loss: 0.7176 - val_accuracy: 0.8333 - val_loss: 0.7066
Epoch 19/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7916 - loss: 0.6430 - val_accuracy: 0.8333 - val_loss: 0.6831
Epoch 20/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8144 - loss: 0.6013 - val_accuracy: 0.8333 - val_loss: 0.6620
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.9000 - loss: 0.5389
Accuracy: 90.0000
```

```
loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
loss = loss * 100
print(f'Test Loss: {loss:.4f}')
accuracy *= 100
print(f'Test Accuracy: {accuracy:.4f}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 395ms/step - accuracy: 0.3000 - loss: 1.3620
Test Loss: 136.2034
Test Accuracy: 30.0000
```

```
model_improved = Sequential([
    Dense(32, input_shape=(X_train.shape[1],), activation='relu'),
    Dense(32, activation='relu'),
    Dense(3, activation='softmax')
])

model_improved.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])

history_improved = model_improved.fit(X_train, y_train, epochs=20, batch_size=10, validation_split=0.1, verbose=1)

loss_improved, accuracy_improved = model_improved.evaluate(X_test, y_test, verbose=1)
accuracy_improved *= 100
print(f'Improved Test Accuracy: {accuracy_improved:.4f}')
```

```
Epoch 1/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 5s 64ms/step - accuracy: 0.7749 - loss: 0.8933 - val_accuracy: 0.7500 - val_loss: 0.9042
Epoch 2/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7370 - loss: 0.7851 - val_accuracy: 0.8333 - val_loss: 0.8228
Epoch 3/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7516 - loss: 0.7180 - val_accuracy: 0.8333 - val_loss: 0.7557
Epoch 4/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7645 - loss: 0.6478 - val_accuracy: 0.8333 - val_loss: 0.6964
Epoch 5/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8068 - loss: 0.5331 - val_accuracy: 0.8333 - val_loss: 0.6461
Epoch 6/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7836 - loss: 0.5038 - val_accuracy: 0.8333 - val_loss: 0.6067
Epoch 7/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8678 - loss: 0.4026 - val_accuracy: 0.8333 - val_loss: 0.5728
Epoch 8/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8777 - loss: 0.3872 - val_accuracy: 0.8333 - val_loss: 0.5425
Epoch 9/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8536 - loss: 0.3873 - val_accuracy: 0.8333 - val_loss: 0.5174
Epoch 10/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8299 - loss: 0.3926 - val_accuracy: 0.8333 - val_loss: 0.4961
Epoch 11/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8399 - loss: 0.3509 - val_accuracy: 0.9167 - val_loss: 0.4783
Epoch 12/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8502 - loss: 0.3487 - val_accuracy: 0.9167 - val_loss: 0.4642
Epoch 13/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8294 - loss: 0.3588 - val_accuracy: 0.9167 - val_loss: 0.4472
Epoch 14/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8824 - loss: 0.2989 - val_accuracy: 0.9167 - val_loss: 0.4338
Epoch 15/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9007 - loss: 0.2612 - val_accuracy: 0.9167 - val_loss: 0.4216
Epoch 16/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8732 - loss: 0.2838 - val_accuracy: 0.9167 - val_loss: 0.4094
Epoch 17/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8861 - loss: 0.2664 - val_accuracy: 0.9167 - val_loss: 0.3957
Epoch 18/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8732 - loss: 0.2764 - val_accuracy: 0.9167 - val_loss: 0.3792
Epoch 19/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8864 - loss: 0.2514 - val_accuracy: 0.9167 - val_loss: 0.3688
```

```
Epoch 20/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9161 - loss: 0.2377 - val_accuracy: 0.9167 - val_loss: 0.3567
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9667 - loss: 0.1868
Improved Test Accuracy: 96.6667
```

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from keras.datasets import mnist
```

```
from keras.datasets import mnist

(train_X, train_y), (test_X, test_y) = mnist.load_data()

print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))
```

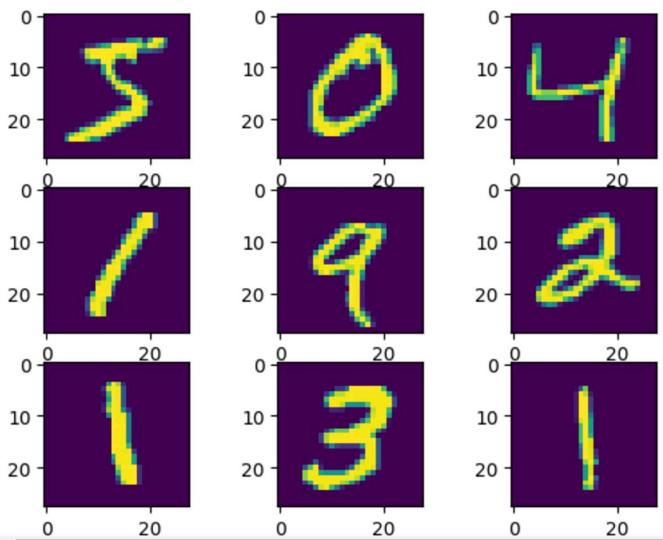
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
print('The shape of the training inputs:', X_train.shape)
print('The shape of the training labels:',y_train.shape)
print('The shape of the testing inputs:',X_test.shape)
print('The shape of the testing labels:',y_test.shape)
```

```
The shape of the training inputs: (60000, 28, 28)
The shape of the training labels: (60000,)
The shape of the testing inputs: (10000, 28, 28)
The shape of the testing labels: (10000,)
```

```
fig, axs = plt.subplots(3, 3)
cnt = 0
for i in range(3):
     for j in range(3):
         axs[i, j].imshow(X_train[cnt])
         cnt += 1
```




```
X_train = tf.keras.utils.normalize(X_train, axis=1)
X_test = tf.keras.utils.normalize(X_test, axis=1)
```

```
model = tf.keras.models.Sequential()
```

```
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(units=128, activation=tf.nn.relu))   # 1st hidden layer
model.add(tf.keras.layers.Dense(units=128, activation=tf.nn.relu))   # 2nd hidden layer
model.add(tf.keras.layers.Dense(units=10, activation=tf.nn.softmax))   # output layer

model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(**kwargs)
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense_9 (Dense)	(None, 128)	100,480
dense_10 (Dense)	(None, 128)	16,512
dense_11 (Dense)	(None, 10)	1,290

```
 Total params: 118,282 (462.04 KB)
 Trainable params: 118,282 (462.04 KB)
 Non-trainable params: 0 (0.00 B)
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=20, batch_size=100)
```

```
Epoch 1/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8302 - loss: 0.6404
Epoch 2/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9541 - loss: 0.1530
Epoch 3/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9692 - loss: 0.1023
Epoch 4/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.9786 - loss: 0.0729
Epoch 5/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9819 - loss: 0.0574
Epoch 6/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9872 - loss: 0.0438
Epoch 7/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9902 - loss: 0.0336
Epoch 8/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9922 - loss: 0.0269
Epoch 9/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9937 - loss: 0.0214
Epoch 10/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9946 - loss: 0.0177
Epoch 11/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9959 - loss: 0.0146
Epoch 12/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9958 - loss: 0.0130
Epoch 13/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9973 - loss: 0.0089
Epoch 14/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9966 - loss: 0.0099
Epoch 15/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9972 - loss: 0.0082
Epoch 16/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9983 - loss: 0.0062
Epoch 17/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9981 - loss: 0.0066
Epoch 18/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9977 - loss: 0.0063
Epoch 19/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.9988 - loss: 0.0043
Epoch 20/20
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9995 - loss: 0.0026
<keras.src.callbacks.history.History at 0x7aa6b9e2d0c0>
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(loss)
accuracy *= 100
print(accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9707 - loss: 0.1477
0.1345250904560089
97.33999967575073
```

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
```

```
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np

# Parameters
max_features = 10000  # Top most frequent words to consider
maxlen = 200  # Max length of review (in words)
embedding_dims = 100  # Dimension of word embedding

print("Loading data...")
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

print("Padding sequences...")
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

print("Building model...")
model = Sequential()
model.add(Embedding(max_features, embedding_dims, input_length=maxlen))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print("Training model...")
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(x_train, y_train,
                    batch_size=32,
                    epochs=15,
                    validation_split=0.2,
                    callbacks=[early_stopping])

print("Evaluating model...")
score, acc = model.evaluate(x_test, y_test, batch_size=32)
print('Test score:', score)
print('Test accuracy:', acc)

# Make predictions on new data
def predict_sentiment(text):
    # Get the word index
    word_index = imdb.get_word_index()

    # Tokenize the new text
    words = text.lower().split()
    new_text = [word_index.get(word, 0) for word in words]

    # Pad the sequence
    new_text = pad_sequences([new_text], maxlen=maxlen)

    # Make prediction
    prediction = model.predict(new_text)
    return "Positive" if prediction[0][0] > 0.5 else "Negative"

# Example usage
sample_review = "This movie was fantastic! I really enjoyed it."
print(f"Sentiment: {predict_sentiment(sample_review)}")
```

```
Loading data...
Padding sequences...
Building model...
Training model...
Epoch 1/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 93s 145ms/step - accuracy: 0.6981 - loss: 0.5607 - val_accuracy: 0.8214 - val_loss: 0.3959
Epoch 2/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 91s 145ms/step - accuracy: 0.8587 - loss: 0.3430 - val_accuracy: 0.7454 - val_loss: 0.5021
Epoch 3/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 91s 145ms/step - accuracy: 0.8499 - loss: 0.3539 - val_accuracy: 0.8256 - val_loss: 0.4170
Epoch 4/15
625/625 ━━━━━━━━━━━━━━━━━━━━ 143s 146ms/step - accuracy: 0.8922 - loss: 0.2720 - val_accuracy: 0.8264 - val_loss: 0.4130
Evaluating model...
782/782 ━━━━━━━━━━━━━━━━━━━━ 30s 39ms/step - accuracy: 0.8218 - loss: 0.3954
Test score: 0.39333289861679077
Test accuracy: 0.8229600191116333
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step
Sentiment: Positive
```

```python
import numpy as np
import tensorflow as tf
import keras
import struct
from array import array
from keras._tf_keras.keras import datasets, layers, models
from os.path import join
import matplotlib.pyplot as plt

# Define file paths for MNIST data files
training_images_filepath = '/content/train-images.idx3-ubyte'
training_labels_filepath = '/content/train-labels.idx1-ubyte'
test_images_filepath = '/content/t10k-images.idx3-ubyte'
test_labels_filepath = '/content/t10k-labels.idx1-ubyte'

# Define the MnistDataloader class (as provided)
class MnistDataloader(object):
    def __init__(self, training_images_filepath, training_labels_filepath,
                 test_images_filepath, test_labels_filepath):
        self.training_images_filepath = training_images_filepath
        self.training_labels_filepath = training_labels_filepath
        self.test_images_filepath = test_images_filepath
        self.test_labels_filepath = test_labels_filepath

    def read_images_labels(self, images_filepath, labels_filepath):
        labels = []
        with open(labels_filepath, 'rb') as file:
            magic, size = struct.unpack(">II", file.read(8))
            if magic != 2049:
                raise ValueError('Magic number mismatch, expected 2049, got {}'.format(magic))
            labels = array("B", file.read())

        with open(images_filepath, 'rb') as file:
            magic, size, rows, cols = struct.unpack(">IIII", file.read(16))
            if magic != 2051:
                raise ValueError('Magic number mismatch, expected 2051, got {}'.format(magic))
            image_data = array("B", file.read())
        images = []
        for i in range(size):
            images.append([0] * rows * cols)
        for i in range(size):
            img = np.array(image_data[i * rows * cols:(i + 1) * rows * cols])
            img = img.reshape(28, 28)
            images[i][:] = img

        return images, labels

    def load_data(self):
        x_train, y_train = self.read_images_labels(self.training_images_filepath, self.training_labels_filepath)
        x_test, y_test = self.read_images_labels(self.test_images_filepath, self.test_labels_filepath)
        return (x_train, y_train), (x_test, y_test)

# Instantiate the dataloader and load the data
mnist_dataloader = MnistDataloader(training_images_filepath, training_labels_filepath, test_images_filepath, test_labels_filepath)
(x_train, y_train), (x_test, y_test) = mnist_dataloader.load_data()

# Convert to numpy arrays
x_train = np.array(x_train)
x_test = np.array(x_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

# Reshape data to add a single channel dimension (grayscale)
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))

# Normalize pixel values between 0 and 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0



model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
# Print model summary
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 64)	102,464
dense_1 (Dense)	(None, 10)	650

Total params: 121,930 (476.29 KB)
Trainable params: 121,930 (476.29 KB)

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train, epochs=50,validation_split=0.1)
```

```
Epoch 1/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 27ms/step - accuracy: 0.9021 - loss: 0.3254 - val_accuracy: 0.9812 - val_loss: 0.0590
Epoch 2/50
```

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc}')
```

```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper left')

plt.show()
```

```
predictions = model.predict(x_test)

# Example: Print the predicted class for the first test image
print("Predicted class for the first test image:", predictions[0].argmax())
print("Actual class for the first test image:", y_test[0])
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from scipy import stats
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding #changed from keras.layers.embeddings to keras.layers
from keras.layers import SimpleRNN,Dense,Activation

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
(X_train,Y_train),(X_test,Y_test) = imdb.load_data(path="imdb.npz",num_words=None,skip_top=0,maxlen=None,
                                                    start_char=1,seed=13,oov_char=2,index_from=3)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
print("Type: ", type(X_train))
print("Type: ", type(Y_train))
```

```
Type:  <class 'numpy.ndarray'>
Type:  <class 'numpy.ndarray'>
```

```
print("X train shape: ",X_train.shape)
print("Y train shape: ",Y_train.shape)
```

```
X train shape:  (25000,)
Y train shape:  (25000,)
```

```
print(X_train[0])
```

```
[1, 608, 13, 6467, 14, 22, 13, 80, 1109, 14, 20, 584, 18, 231, 72, 141, 6, 783, 254, 189, 7060, 13, 100, 115, 106, 14, 20, 584, 207,
```

```
review_len_train = []
review_len_test = []
for i,j in zip(X_train,X_test):
    review_len_train.append(len(i))
    review_len_test.append(len(j))
```

```
print("min: ", min(review_len_train), "max: ", max(review_len_train))
```

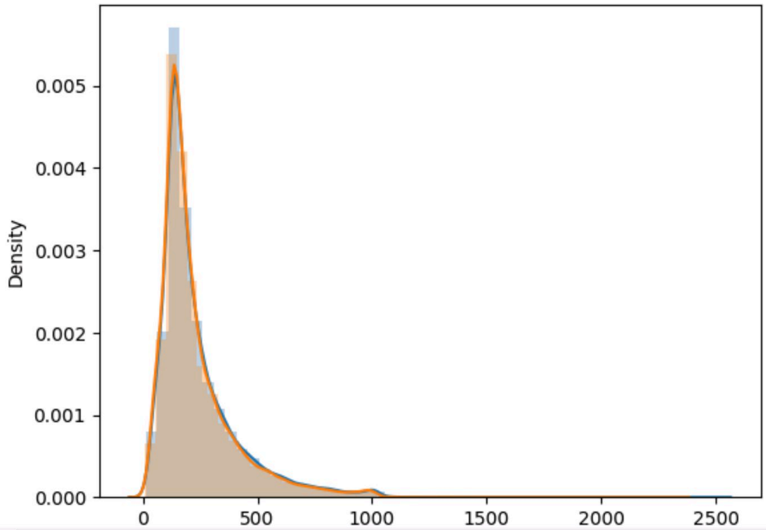
```
min:  11 max:  2494
```

```
print("min: ", min(review_len_test), "max: ", max(review_len_test))
```

```
min:  7 max:  2315
```

```
sns.distplot(review_len_train,hist_kws={"alpha":0.3})
sns.distplot(review_len_test,hist_kws={"alpha":0.3})
```

```
<ipython-input-9-7037aabe6f55>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(review_len_train,hist_kws={"alpha":0.3})
<ipython-input-9-7037aabe6f55>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(review_len_test,hist_kws={"alpha":0.3})
<Axes: ylabel='Density'>
```




```
print("Train mean: ",np.mean(review_len_train))
print("Train median: ",np.median(review_len_train))
print("Train mode: ",stats.mode(review_len_train))
```

```
Train mean:  238.71364
Train median:  178.0
Train mode:  ModeResult(mode=132, count=196)
```

```
# number or words
word_index = imdb.get_word_index()
print(type(word_index))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
<class 'dict'>
```

```
def whatItSay(index=24):
    reverse_index = dict([(value,key) for (key,value) in word_index.items()])
    decode_review = " ".join([reverse_index.get(i-3, "!") for i in X_train[index]])
    print(decode_review)
    print(Y_train[index])
    return decode_review

decoded_review = whatItSay()
```

```
! this movie was extremely funny i would like to own this for my vintage collection of 1970s movie must see again list i know this c
1
```

```
decoded_review = whatItSay(5)
```

```
! quite possibly how francis veber one of the best comedy directors in the world at least when sticking to his native france managec
0
```

```
num_words = 15000
(X_train,Y_train),(X_test,Y_test) = imdb.load_data(num_words=num_words)
```

```
maxlen=130
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

```
print(X_train[5])
```

```
[    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    1  778  128   74   12  630  163   15    4 1766 7982
  1051    2   32   85  156   45   40  148  139  121  664  665   10   10
  1361  173    4  749    2   16 3804    8    4  226   65   12   43  127
    24    2   10   10]
```

```
rnn = Sequential()

rnn.add(Embedding(num_words,32,input_length =len(X_train[0]))) # num_words=15000
rnn.add(SimpleRNN(16,input_shape = (num_words,maxlen), return_sequences=False,activation="relu"))
rnn.add(Dense(1)) #flatten
rnn.add(Activation("sigmoid")) #using sigmoid for binary classification

print(rnn.summary())
rnn.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accuracy"])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. :
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argur
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
simple_rnn (SimpleRNN)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
activation (Activation)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
None

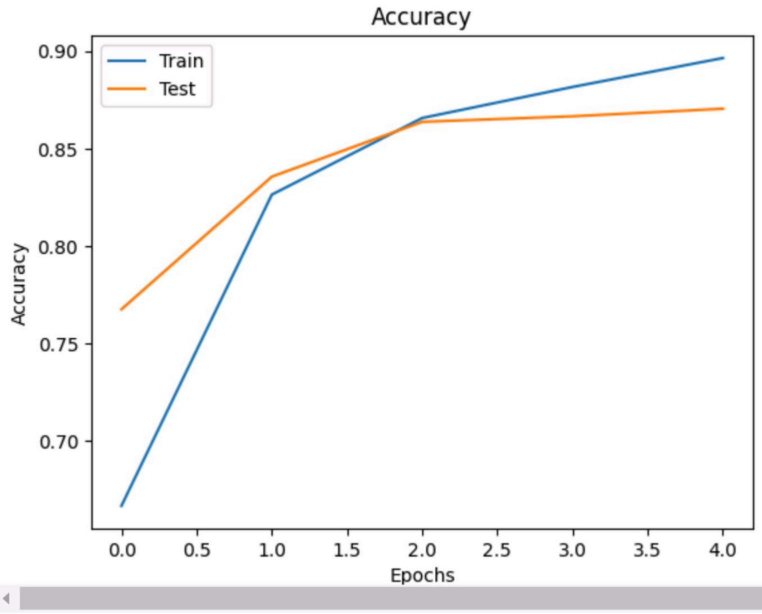
```
history = rnn.fit(X_train,Y_train,validation_data = (X_test,Y_test),epochs = 5,batch_size=128,verbose = 1)
```

```
Epoch 1/5
196/196 ━━━━━━━━━━━━━━━━━━━━ 13s 51ms/step - accuracy: 0.5938 - loss: 0.6658 - val_accuracy: 0.7676 - val_loss: 0.4872
Epoch 2/5
196/196 ━━━━━━━━━━━━━━━━━━━━ 11s 54ms/step - accuracy: 0.8173 - loss: 0.4257 - val_accuracy: 0.8356 - val_loss: 0.3813
Epoch 3/5
196/196 ━━━━━━━━━━━━━━━━━━━━ 19s 48ms/step - accuracy: 0.8623 - loss: 0.3308 - val_accuracy: 0.8637 - val_loss: 0.3256
Epoch 4/5
196/196 ━━━━━━━━━━━━━━━━━━━━ 9s 48ms/step - accuracy: 0.8834 - loss: 0.2804 - val_accuracy: 0.8665 - val_loss: 0.3170
Epoch 5/5
196/196 ━━━━━━━━━━━━━━━━━━━━ 10s 51ms/step - accuracy: 0.9002 - loss: 0.2591 - val_accuracy: 0.8704 - val_loss: 0.3118
```

```
score = rnn.evaluate(X_test,Y_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.8705 - loss: 0.3121
```

```
plt.figure()
plt.plot(history.history["accuracy"],label="Train");
plt.plot(history.history["val_accuracy"],label="Test");
plt.title("Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```




```
plt.figure()
plt.plot(history.history["loss"],label="Train");
plt.plot(history.history["val_loss"],label="Test");
plt.title("Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
plt.show();
```